PHP logic & loops

CIS 1152 Adv Web Dev

Lecture 3

Steve Ruegsegger

Outline

Goal: the *very basics* of PHP scripting

Objectives:

1. Stateless
2. Conditional branching
   - If Then Else
   - Logical operations
   - Switch (case)
3. unset/missing/null vars -- these are problematic
4. Loops (3 types)
5. Nested loops

What is stateless?

- PHP (default) is **stateless**
- What does this mean?
  - Discussed in class!
  - Hint: does PHP 'remember' the variable values every time you refresh the browser?
- Demo... *l03_stateless1* and *l03_stateless2*
- How to "fix" – i.e. how to remember 'states'?
  Stay tuned...

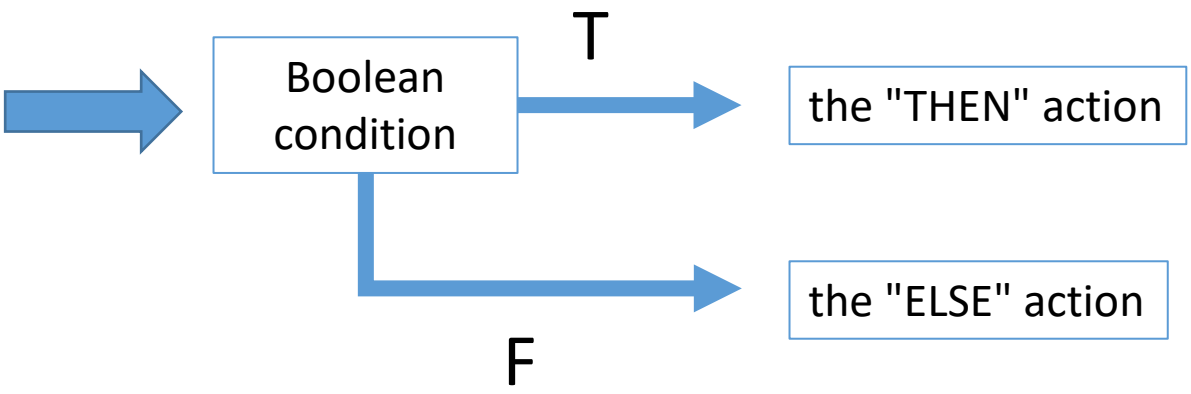
Conditionals

i.e. "branching"

The Missing Link -- ch 31, p179

Branching

- Make a decision! Take path A or path B.
- *How* to choose which branch/path?
  Boolean logic → True / False
- Flow diagram:



Conditionals

```
if (Boolean condition) {
        // True statements
}
else {
        // False statements
}
```

Comparison / Boolean Operators

Operator	Name	Use
==	Equals	$a == $b
===	Identical	$a === $b
!=	Not equal	$a != $b
!==	Not identical	$a !== $b
<>	Not equal (comparison operator)	$a <> $b
<	Less than	$a < $b
>	Greater than (comparison operator)	$a > $b
<=	Less than or equal to	$a <= $b
>=	Greater than or equal to	$a >= $b

The first 2 are tricky:
(next page)

Equals vs Identical

- **Equals**, with 2 ==, does <u>not</u> consider variable **type**.
  - 0, NULL, unset, empty are all the same
  - 1, True, non-empty are the same
- **Identical**, with 3 ===, does consider variable **type**.

l03_equal_identical

```
8   $answer = "no";
9   echo("<p>\$answer is set to '$answer'.<br><br>");
10
11  # is there a "y" in the answer?
12  # returns FALSE or the integer position
13  $yes = strpos($answer, "y");
14  echo("<p>\$yes from strpos() is '$yes'.<br><br>");
15
16  if ($yes >= 0) { echo ("positive answer"); }
17  else { echo("negative answer"); }
18
19
```

L3a equal vs identical

$answer is set to 'no'.

$yes from strpos() is ''.

positive answer

Only **2** chars is *equals* . It takes **3** chars for **identical**

Complex conditions

```
if (condition_1 AND condition_2) {
        // True statements
}
else {
        // False statements
}
```

Logical Operators

Operator	Name	Use	Result
!	NOT	!$b	Returns true if $b is false and vice versa
&&	AND	$a && $b	Returns true if both $a and $b are true; otherwise false
||	OR	$a || $b	Returns true if either $a or $b or both are true; otherwise false
and	AND	$a and $b	Same as &&, but with lower precedence
or	OR	$a or $b	Same as ||, but with lower precedence
xor	XOR	$a x or $b	Returns true if either $a or $b is true, and false if they are both true or both false.

If Then Else example

```
$cost=43;          # how much I spent
$limit = 100;  // what the boss says

if ($cost <= $limit) {
      print 'This is a legal expenditure!';
      $result = 'ok';
}
else {
      print 'Uh-oh! You are in trouble';
      print "$cost is too much";
      $result = 'You are fired!';
}
echo "The result:  $result<br>";
```

Variable type tests

- Value status – these return T or F
- Remember these tests:
  - `is_int(), is_float(), is_string(), is_bool()`
  - `is_null(), is_scalar(), is_array(), is_numeric()`

```
if is_numeric( $input ) {
        echo "Your input is: $input <br>";
} else {
        echo "Warning, you need to give a number";
}
```

If-then "shortcut" format

- Cool programming shortcut
- From C-language: Called the "ternary operator" A ? B : C
- Syntax:

```
<condition> ? <value if true> : <value if false>
```

- Examples:

```
$sale =  $offer > $price ? 'Y' : 'N;
$movie = ( $age < 13 ) ? 'G' : 'PG' ;
```

- Used a lot:
  - To see if a key is in an array (more later)

If elseif... else template

```
if (condition1) {
   // if c1 is true
}
elseif (condition2) {
   // if c2 is true
}
elseif (condition3) {
   // if c3 is true
}
else {
   // if everything above is false, then do this
}
```

Example: If elseif... else

```
$price = 100;
// discount in percentage units
if ($quantity < 10) {
    $discount  = 0.0;  }
elseif ($quantity < 20) {
    // between 10 and 20-
    $discount  = 5.0;  }
elseif ($quantity < 30) {
    // between 20 and 30-
    $discount  = 10.0;  }
else {
    // whoa, more than 30
    $discount  = 20.0;  }

$total = $price * (1 - $discount/100);
echo "Your bill is \$ $total";
```

Style note #1

Where do the curly brackets go?

1970's → your prof

```
<?php
    $ans = 42;
    if ( $ans == 42 ) {
        print "Hello world!\n";
    } else {
        print "Wrong answer\n";
    }
?>
```

Aesthetics

the "new" way...

```
<?php
    $ans = 42;
    if ( $ans == 42 )
    {
        print "Hello world!\n";
    }
    else
    {
        print "Wrong answer\n";
    }
?>
```

Style note #2

- {}'s are actually "optional"
- Curly brackets mean "block of multiple statements".
- If you only have 1 statement, then you don't need curly brackets!

```
if     ($page == "Home")  echo "You selected Home";
elseif ($page == "About") echo "You selected About";
elseif ($page == "News")  echo "You selected News";
elseif ($page == "Login") echo "You selected Login";
elseif ($page == "Links") echo "You selected Links";


if     ($page == "Home")  { echo "You selected Home"; }
elseif ($page == "About") { echo "You selected About"; }
elseif ($page == "News")  { echo "You selected News"; }
elseif ($page == "Login") { echo "You selected Login"; }
elseif ($page == "Links") { echo "You selected Links"; }
```

switch

- Helpful for long if-then-elseif-...-elseif-else conditions

```
switch ($var) {
    case "value1" :
            // do this if $var == value1
            break;
    case "value2" :
            // do this if $var == value2
            break;
    // lots more
    default:
            // otherwise, do this
            break;
}
```

switch

- Switch compares the initial value to the other cases with an == operator.
- Mostly used with strings.

```
switch ($user) {
    case "bob"  : $permission = 'R';   break;
    case "joe"  : $permission = 'RW';  break;
    case "fred" : $permission = 'RWX'; break;
    default: $permission =''; break;
}
echo "$user is allowed to <b>$permission</b> <br>";
```

switch

- Previous discount example.
- Numerics are a little tricky. Set the *initial* switch value to True, then have Booleans, and the first true wins!

```
$price = 100;
switch (true) {
    case ($quantity < 10) : $discount = 0; break;
    case ($quantity < 20) : $discount = 5; break;
    case ($quantity < 30) : $discount = 10; break;
    default: $discount = 20; break;
}
$total = $price * (1 - $discount/100);
echo "Your bill is \$ $total";
```

unset/missing/null

These are *problematic*

Unset vars

Important note:

- PHP does not like *un*initialized, *un*set vars!
- It will *fuss* at you.
- Often, variables from a HTML form are left empty and therefore not in $_POST and therefore uninitialized...
- What to do? ...

How to "deal" with possible unset vars

- Just a var:

```
if ($var) {
    // true is $VAR is NOT: 0, "", FALSE, or NULL
    // what if it is undefined???  → ERROR
```

- is_*($var)

```
if ( is_numeric($var) ) {
    // true if $var is a number
```

Problem

Solution

1. Use empty()

```
if ( empty($var) ) {
    // true if $var is 0, "", FALSE, or NULL
    // with NO FUSSING
```

2. Use isset()

```
if ( isset($var) ) {
    // true if $var has any value other than NULL
    // (including 0, FALSE or "")
```

Loops

while loop

```
$k=1;
while ($k < 100)  {
      print $k;
      $k=2*$k;    // double k
}
```

Boolean expression: T/F
"Keep looping as long as T"

*There must be an expression inside the loop to change the **iteration variable**!*

- This is called a "*zero-trip loop*" . It *might* not run.
- $k is the *iteration variable*
- The T/F expression is evaluated first and the loop may or may not run.
- Note: this example looks really poor in HTML. Can you make this example code look much better in a web browser?

do...while loop

- This is called a "*one trip loop*". It will always run at least 1 time.
- Just like the previous while loop, but...
  the condition is at the end of the loop *iteration*.
- $i is the iteration variable.
- We programmer must manage/change the iteration variable in the loop!

```
$i=10;
do {
   print "this is loop $i <br>";
   $i--;
} while ($i > 0);
```

For loop

- C-style for loop.
- A "numbered-loop".
- 3 expressions...
- Can you describe each one and where they "go" in the loop?

```
# multiplication table
$m=8;
$end = 10;
for ($n=1; $n<=$end; $n++) {
    $ans = $m * $n;
    print "$n * $m = $ans <br>";
}
```

Interrupt loop flow

- There are 2 ways to *interrupt* loop flow
  - `break` – stop this loop structure. No more looping. Move to next command.
  - `continue` – stop this *specific* loop iteration. Go to *top* and start over with the next loop iteration.

```
for ($m=1; $m<=40; $m++) {
    if ($m % 2) continue;  # skip odd numbers
    print "now m is ".$m . "!<br>";
}
```

The forever loop!

- What if you don't know when the loop will end?
- Use an infinite loop and break; when a condition is met

```
while (TRUE) {
    // process some info here...
    if $value >= $max {
       break;
    }
}
```

The *understood* else

- Often, I'll use an "if" branch to look for a condition and then break, or exit or continue.
- The lines of code after that if-then are "*by definition*" the "*else*" part of the if-then-else branch.
- But I didn't have to write the else nor indent the code!

```
while (TRUE) {
    // process some info here...
    if ($value >= $max) {
       break;
    }
    # These lines are, by definition, the else.
    # If you get here, then you did not break;
    # Therefore, I know $value must be < $max!
    print("Your value $value is valid and < $max\n");
}
```

Looping over chars in a string

- We often need to *parse* input strings.
- String functions to memorize:
  - $n = strlen($inputString); # how many chars in the string
  - $c = $inputString[$i]; # $c is the ith char in the string
- If we want to loop over each char in string:

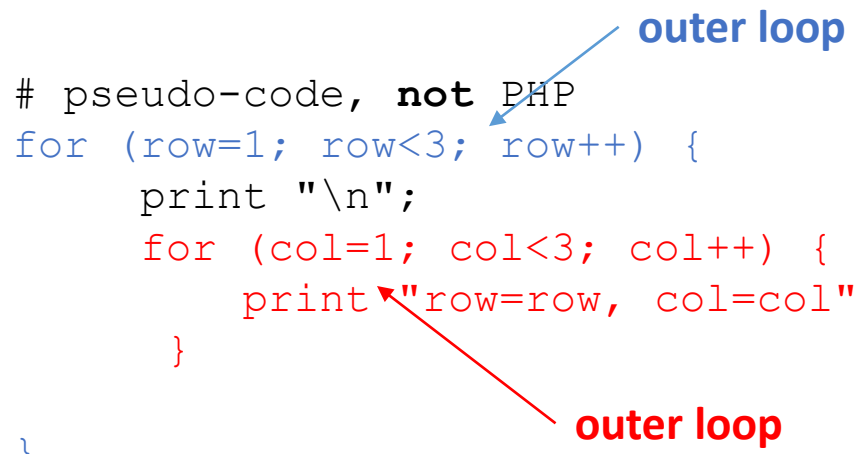
```
# use C-style for loop
echo("string = $inputString <br>");
echo("<ul>");
for ($i=0; $i < strlen($inputString); $i++) {
    echo("<li>char #$i = $inputString[$i]");
}
echo("</ul>");
```

Nested Loops

Creating HTML tables from PHP loops...
we do this all the time

A loop within a loop

- Within an "outer" loop -- run another entire "inner" loop
- common examples:
  - hierarchical data:
    - address *per* student
    - items in carts
  - 2-D data
    - multiple temps per day
    - cols within a row

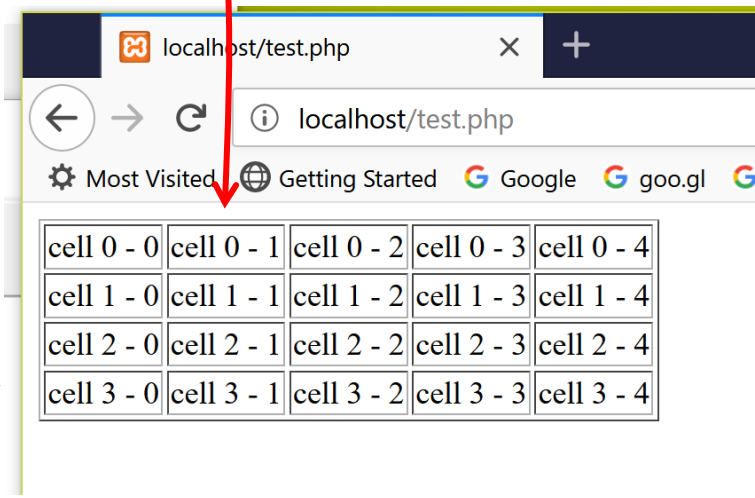



```
row = 1, col=1
row = 1, col=2
row = 1, col=3

row = 2, col=1
row = 2, col=2
row = 2, col=3

row = 3, col=1
row = 3, col=2
row = 3, col=3
```

HTML tables with nested loops

```
print "<table border=1>";
$rows = 4;   # rows in table
$cols = 5;   # cols per row
$r=0;
while ($r < $rows){
    print "<tr>";  # new row
    $c=0;
    while ($c < $cols) {
        # each cell in this row
         print "<td>cell $r - $c</td>";
         $c=$c+1;
    }
    print "</tr>\n"; # end of row
    $r=$r+1;
}
print "</table>\n\n";
```

write rows of cols →



cell 0 - 0	cell 0 - 1	cell 0 - 2	cell 0 - 3	cell 0 - 4
cell 1 - 0	cell 1 - 1	cell 1 - 2	cell 1 - 3	cell 1 - 4
cell 2 - 0	cell 2 - 1	cell 2 - 2	cell 2 - 3	cell 2 - 4
cell 3 - 0	cell 3 - 1	cell 3 - 2	cell 3 - 3	cell 3 - 4

l03_tables_while

Lab 2 this week

We will use PHP loops and conditionals to make fancy HTML tables and game boards.